

A Faster Hardware Implementation of RSA Algorithm

Ajay C Shantilal

Department of Electrical & Computer Engineering,
Oregon State University, Corvallis, Oregon 97331 -USA.
E-mail: ajay@ece.orst.edu

Abstract—The performance of most crypto systems is primarily determined by an efficient implementation of arithmetic operations. When implementing public key cryptography such as RSA the primary requirements are high speed arithmetic computation, small size and low power consumption and resistance to side channel attacks. In this paper an efficient way to explore fast modular operation has been explored, using redundant digit sets with higher radices and making modifications to Montgomery's Algorithm in order to explore deep pipelining at architecture level which improves the throughput and latency of the system. This paper presents a solution to the problems of existing methods, proposes an actual implementation of the solution and demonstrates the benefit of the proposed approach. I suspect my algorithm to be nearly optimal and challenge the cryptographic community for better results.

Keywords: RSA, modular exponentiation, modulo multiplication, bit serial multiplier architecture, pipelining, redundant digit set, RNS, Chinese remainder theorem power analysis, timing attacks.

I. INTRODUCTION.

It is widely recognized that security issues will play a crucial role in the majority of future computer and communication systems. Central tools for achieving system security are cryptographic algorithms. For performance as well as for physical security reasons it is often required to realize cryptographic algorithms in hardware. Computational performance of large integers is important in implementation of public key cryptography. From mathematical viewpoint the cryptographic algorithms have a common characteristics: they perform long integer modular exponentiation. Hence to improve the performance of cryptographic system it is required to fasten the modular exponentiation. Many papers have been proposed in this direction but they are a trade off between area and power consumption and are vulnerable to side channel attacks.

This paper proposes a scheme, which takes into consideration area, and possible side channel attacks while improving the performance of modular exponentiation significantly. In this paper I implemented RSA algorithm suitable for DSP and can be extended to any system. For Modular multiplication I devised a fast implementation method for Montgomery multiplication, which explores deep and parallel pipelining. Montgomery multiplication is based on Redundant Number System (RNS). In RNS an integer is

represented by set of its residues in terms of base elements of RNS, and thus arithmetic operations can be independently carried out for every base element. On the other hand Montgomery multiplication is a method for performing modular multiplication by substituting additions and multiplication for division. Therefore the combination of RNS and Montgomery is expected to be well suited for parallel processing of modular exponentiation [1]. I also used Chinese Remainder theorem (CRT) to improve the performance. Further within each processing unit high radix digit representation (radix - 16) is used to reduce the number of clock cycles. I extend double modular exponentiation to resist side channel attacks that extracts secret exponent by analyzing the target power consumption and timing analysis at negligible overhead.

This contribution is structured as follows. In Section 2, I summarize some of the previous work on modular exponentiation. Section 3 describes algorithms for modular exponentiation and architecture to implement the same. Section 4 of this contribution shows the timing and area results obtained and makes a comparison of our results to previous work. In section 5 conclusions are drawn based on performance, area, data integrity and side channel attacks.

II. PREVIOUS WORK

In the following I will summarize relevant previous work in field of modular multiplication. Most presented approaches are based on the algorithm proposed by Peter Montgomery [2] either in conjunction with redundant number system or systolic array architecture. In [3] right to left binary exponentiation algorithm was used for double modular exponentiation. This scheme has a low power consumption and is resistant to side channel attacks but do not have any gain on the performance. In [4] the author focuses on the security gains related to power attacks against smart cards but have short coming that it can not be implemented if the size of the modulus is smaller than the size of the multiplier and we have to pre compute $2^{2^{pt}} \bmod N$. In [5] the author implements and combines a high radix version of Montgomerys algorithm with a novel systolic array architecture and concludes that the performance are better than previously reported implementations on radix 2. It also had a significant gain in the area but is vulnerable to timing and power attacks. In reference [6] the authors have implemented a method for Montgomery's multiplication which is useful for pipeline processing and have devised

an improved computation for the number of multiplications and additions. They do not give any details about the area and the power consumption with their method. In [7] this the author implements Montgomery multiplication in systolic array method. He also gives solutions to bottlenecks arising in hardware for implementing RSA for classical algorithm but however suffers from broadcasting problems with classical algorithm and scheduling complications with Montgomery's Algorithm. In [8] the author explores Barrets Modular Reduction Method instead of Montgomery to avoid the division in Modular multiplication. It has a high degree of parallelism in the Multiplier code which is the most significant characteristic of the proposed scheme. It also explores Chinese Remainder theorem to improve the performance at the cost of extra area and higher power consumption.

III. RNS BASED HIGH RADIX MM

A. Residue Number System

In RNS an integer 'x' is represented by

$$\langle x \rangle_a = (x[a_1], x[a_2], \dots, x[a_n])$$

where $x[a_i] = x \bmod a_i$. The set $a = a_1, a_2, \dots, a_n$ is called a base and the number of elements n in it is base size, where a_i 's are mutually prime. Due to Chinese remainder theorem any 'x' which satisfies $0 \leq x \leq A (A = \prod_{i=1}^n a_i)$ has one and only one RNS representation.

Addition and multiplication modulo 'a' can be implemented in parallel in linear space ($\lg(n)$ channels) and [performed in one single step without any carry propagation as follows.

$$\begin{aligned} \langle x \pm y \rangle_a &= ((x[a_1] \pm y[a_1])[a_1], \dots, (x[a_n] \pm y[a_n])[a_n]) \\ \langle x \cdot y \rangle_a &= ((x[a_1] \cdot y[a_1])[a_1], \dots, (x[a_n] \cdot y[a_n])[a_n]) \end{aligned}$$

The combination of RNS and MM is expected to realize fast parallel processing effectively.

Montgomery Multiplication Algorithm MM is known to be an efficient method for implementing modular exponentiation in public key cryptography. The Montgomery algorithm computes $MonPro(x, y) = x \cdot y \cdot r^{-1} \bmod n$. given $x, y < n$ and r such that $GCD(n, r) = 1$. Even though the algorithm works for any r, which is relatively prime to n it is more useful when r is taken to be a power of 2, which is intrinsically fast operation on general purpose computer example signal processors. The version shown below taken from [6].

Function: Montgomery Multiplication Input : x, y, N, r

Output: $w = x \cdot y \cdot r^{-1}$

- 1: $t = x \cdot y$
- 2: $m = (t \bmod r)N' \bmod r$
- 3: $t = t + m \cdot n$
- 4: $w = t/r$

B. RNS Montgomery Multiplication with higher radices;

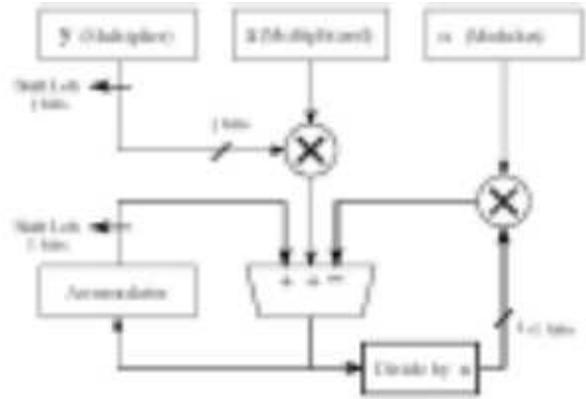
Since the speed for radix -2 multipliers is approaching limits, the use of higher radices is proposed. High radix op-

erations require fewer clock cycles, however the cycle time and area increases but effectively efficiency improves significantly. Let 2^l be the radix. The key operation in computing $w = x \cdot y \pmod n$ is the computation of

$$w = 2^l \cdot w + x_i y_i - Q \cdot n$$

Where w is the partial product and y_i is the i^{th} digit of y in radix 2^l . The value Q determines the number of times the modulus N is subtracted from the partial product w in order to reduce it modulo n. we compute Q by dividing the current value of partial product w by n, which is then multiplied by n and subtracted from partial product during the next cycle. This implementation is illustrated in following figure 1.

Figure 1: High Radix Modular Multiplication



The partial product generation is much more complex for higher radices. However the generation of high radix partial product does not greatly increase cycle time since this computation can be easily pipelined. The most complicated step is the reduction step, which necessitates more complex routing increasing chip area. In [1] has shown an effective modular reduction technique which is being used here.

The above mentioned MM procedure is rewritten using RNS with higher radix as shown below.

Function: RNS MM

Input : $\langle x \rangle_{(aub)}, \langle y \rangle_{(aub)}, (x, y < 2N)$

Output: $\langle w \rangle_{(aub)} (w \equiv x \cdot y \cdot (B^{-1}) \bmod N, w < 2N)$

- 1a: $\langle t \rangle_a = \langle x \cdot y \rangle_a$
- 1b: $\langle t \rangle_b = \langle x \cdot y \rangle_b$
- 2: $\langle m \rangle_b = \langle t \cdot N' \rangle_b$
- 3: $\langle m \rangle_a = BT(\langle m \rangle_b, 0)$
- 4: $\langle u \rangle_a = \langle tN \rangle_a$
- 5: $\langle v \rangle_a = \langle t + u \rangle_a$
- 6: $\langle w \rangle_a = \langle vB^{-1} \rangle_a$
- 7: $\langle w \rangle_b = BT(\langle w \rangle_a, 0.5)$

Here BT is nothing but base transformation which can be implemented using base transformation algorithm in parallel as shown in [1].

An exponentiation algorithm based on radix - 16 is realized by the RNS Montgomery multiplication as shown in the algorithm below. It is assumed that an input variable has been transformed previously into $x' = xB \bmod N$, because of the essential feature of MM in which Montgomery constant B is introduced [1]. From this assumption, the output $y = x^d B \bmod N$ is realized.

Function: Modular Exponentiation using RNS MM

Input : $\langle x \rangle_{(aub)}, d = (d_k, \dots, d_1)_{(2^4)}$

Output: $\langle y \rangle_{(aub)}, s.t. y = x^d B^{-1} \bmod N$

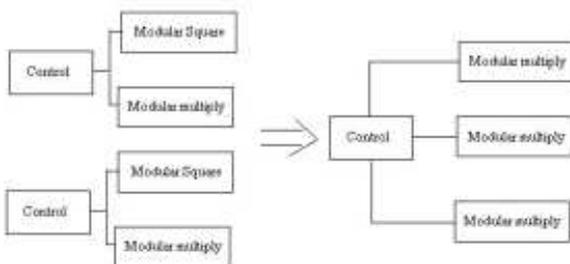
- 1: $\langle x_N^0 \rangle = \langle B_N \rangle$
- 2: $\langle x_N^1 \rangle = \langle x \rangle$
- 3: $\langle x_N^{i+1} \rangle = MM(\langle x_N^i \rangle, \langle x \rangle, N)$
- 4: $\langle y \rangle = \langle x_N^{d_k} \rangle$
- 5: For $i = k - 1, \dots, 1$
- 6: For $j = 1, \dots, 4$
- 7: $\langle y \rangle = MM(\langle y \rangle, \langle y \rangle, N)$
- 8: Next j
- 9: $\langle y \rangle = MM(\langle y \rangle, \langle x_N^{d_i} \rangle, N)$
- 10: Next i

The number of clocks to perform the RNS MM using high radix are $\ln(n^2/ul)$ where u is the number of parallel processing units and l is the radix power of operation. The performance of RNS modular exponentiation is estimated by $\ln(n^3/ul)$. The chip size depends upon number of parallel processing unit, which is determined by u.

C. Double modular exponentiation to resist side channel attacks:

Any circuit to be resistant to power analysis attacks must operate with constant current variation and calculation time regardless of input values. I use a "compensating calculation" forcing the operation to always perform both a modular square circuit and a modular multiply. Thus every circuit is constantly operated even if the exponent is "0". By adjusting the calculation time to give double time for "0" bits the circuit can also be made resistant to timing attacks. But while doing this the area increases considerably shown below is a technique that can overcome this problem by combining two separate modular multiply units into one for shared use. figure[2] illustrates how this can be achieved.

Figure 2: Compensating Circuit Configuration



In the new circuit configuration, the shared modular multiply unit in the center shown above cannot be used for double modular exponentiation calculation when both exponents are equal to "1". In this case, one of the calculations may have to be delayed until the modular multiply unit becomes idle. Proper scheduling and control can reduce this delay. Also we can extend on-fly canonical booth recoding to have more number of "0" in the operand.

IV. TIMING AND AREA RESULTS

In order to prove my algorithm to be the most efficient way ever possible I implemented the above algorithm in VHDL and various timing analysis were carried. Once the design was developed in VHDL, boolean logic and various timing errors were verified by simulating the gate level description with edifout VHDL analyzer. The next step involved the synthesis of the VHDL code with elsyn and obtained the area and delay analysis for ideal condition also obtained the actual gate level delay in sdf format then the VHDL was recompiled using the above obtained actual gate report with routing delays and analyzed for timing and gate delays in XILINX format. Once all the possible delays were incorporated the final design was used to obtain the results shown below using Xilinx timing analyzer and finally verified with an actual chip.

Approach used	RadixUsed	CLB's	t(ms)
RNS based MM	2	5450	0.46
RNS based MM	16	6286	0.12
Double Exponentiation	2	5950	0.56
Double Exponentiation	16	6686	0.14

Table 1: Application to RSA : Encryption.

I compared my fastest RSA 1024 bit design of table 1 to the fastest hardware solutions found in the literature [5]. The proposed method has an encryption time of 0.14 ms which is about 35% faster than the 1024 hardware implementation (.22 ms with radix 16 and .75 ms with radix 2) on a 150 MHz Alpha [5]. I did not find any other system faster than my proposed scheme and suspect my algorithm to be the most optimal one and challenge the cryptographic community for better results. My system has a slightly greater area and power consumption, approximately 8% but is resistant to power attacks and timing attacks which I feel are the most important features for any cryptosystem, which do not exist in the system with which I make my comparison. Systems which possess this feature are far lagging in comparison to performance with my algorithm. The proposed algorithm is about 20 times faster to any other algorithm that has resistance to side channel attacks.

V. CONCLUSIONS

In this paper, I have introduced a new efficient technique for multiplying and exponentiating of arithmetic operation using Residue number system with high radix along with Montgomery multiplication. Also double exponentiation

method was extended in order to make the methodology resistant to side channel attacks. We prove our case by implementing our algorithm and verifying it with the best known algorithm and found that it was 35% faster. If modular multiplication is implemented on processors which do not support arithmetic on full data width, I believe that an RNS implementation is preferable. We can exploit more parallelism using more processors but care has to be taken about the associated area with it.

REFERENCES

- [1] A. Shimbo H. Nozaki, M. Motoyama and S. Kawamura, "Implementation of rsa algorithm based on rns montgomery multiplication," in *Cryptographic Hardware and Embedded Systems - CHES 2001*, C. Paar (Eds.) .K. Ko, D. Naccache, Ed. 2001, pp. 364–376, Springer, Berlin, Germany.
- [2] P. Montgomery, "Modular multiplication without trial division," in *Mathematics of Computations, Volume - 44*, 1985, pp. 519–521.
- [3] Jun Anzai Takehiko Kato, Satoru Ito and Natsume Matsuzaki, "A design for modular exponentiation coprocessor in mobile telecommunication terminals," in *Cryptographic Hardware and Embedded Systems - CHES 2000*, C. Paar (Eds.) .K. Ko, Ed. 2000, pp. 216–228, Springer, Berlin, Germany.
- [4] Gal Hachez and Jean-Jacques Quisquater, "Montgomery exponentiation with no final subtractions: Improved results," in *Cryptographic Hardware and Embedded Systems - CHES 2000*, C. Paar (Eds.) .K. Ko, Ed. 2000, pp. 293–301, Springer, Berlin, Germany.
- [5] Christof Paar Thomas Blum, "High radix montgomery modular exponentiation on reconfigurable hardware," in *Cryptography and Information Security Research Laboratory - CRIS 2001*, 2001, pp. 759–764.
- [6] Naoya Torii Syouji Temma Kouichi Itoh, Masahiko Takenaka and Yasushi Kurihara, "Fast implementation of public-key cryptography on a dsp tms320c6201," in *Cryptographic Hardware and Embedded Systems - CHES 1999*, C. Paar (Eds.) .K. Ko, Ed. 1999, pp. 61–72, Springer, Berlin, Germany.
- [7] Colin D. Walter, "Montgomery's multiplication technique: How to make it smaller and faster," in *Cryptographic Hardware and Embedded Systems - CHES 1999*, C. Paar (Eds.) .K. Ko, Ed. 1999, pp. 80–93, Springer, Berlin, Germany.
- [8] Johann Groschdl, "High-speed rsa hardware based on barret's modular reduction method," in *Cryptographic Hardware and Embedded Systems - CHES 2000*, C. Paar (Eds.) .K. Ko, Ed. 2000, pp. 191–203, Springer, Berlin, Germany.